APPENDIX A: Address Bus Arbitration Algorithm

```
function [0:4] CalcBG;

        input [0:4] BR;
        input [0:4] OldBG;
        input       ParkMode;
        input [0:2] ParkVal;

        if (&BR) begin
              if (!ParkMode) CalcBG = OldBG;
              else case (ParkVal)                    // synopsys full_case parallel_case
                    2'b000: CalcBG = 5'b01111;       // VIDEO
                    2'b001: CalcBG = 5'b10111;       // EXPANSION 1
                    2'b010: CalcBG = 5'b11011;       // EXPANSION 2
                    2'b011: CalcBG = 5'b11101;       // CPU0
                    2'b100: CalcBG = 5'b11110;       // CPU1
              endcase
        end else casex (BR)                          // synopsys full_case parallel_case
              5'b0xxxx: CalcBG = 5'b01111;           // VIDEO
              5'b10xxx: CalcBG = 5'b10111;           // EXPANSION 1
              5'b110xx: CalcBG = 5'b11011;           // EXPANSION 2
              5'b1110x: CalcBG = 5'b11101;           // CPU0
              5'b11110: CalcBG = 5'b11110;           // CPU1
        endcase

endfunction
```

APPENDIX B: Data Bus Arbitration Algorithm

(a pseudo-code summary is more appropriate here)

- if at least one master queue is non-empty
  - select the highest priority non-empty master queue, based upon the following priority encoding:
    - 0: VIDEO (Highest)
    - 1: EXPANSION 1
    - 1: EXPANSION 2
    - 2: CPU 0
    - 3: CPU 1 (Lowest)
  - upon examining the selected master queue to see which slave is selected in the front entry, look at the front entry of the associated slave queue to see if it points back to the selected master. If it does, a master/slave match occurs.
  - if a master/slave match has occurred, grant the data bus to the selected master (via DBG) and slave (via SSD). Otherwise, remain idle.
- otherwise remain idle

Appendix C: Retry generation

The first term, L2Retry, will kill a transaction that the cache cares about if that master already has an outstanding transaction to an expansion bridge. CPU writes to memory are an exception to this rule, since we'll use DBWC.

The second term, TransFullRetry, kills an access when we already have the maximum number of outstanding transactions (3).

The third term, ExRdRetry, is an OR of a vector showing an AAck of a master that has an outstanding expansion bridge read. This transaction must be to a non-bridge slave, and only writes to memory are excepted.

The fourth term, ExCrossRdRetry, will kill an expansion bridge's master read of the other expansion bridge if the expansion bridge master already has a slave read outstanding to it, Or if the expansion bridge slave already has an outstanding read.

The fifth term, ExWrRetry, will kill a CPU write to an expansion bridge that has an outstanding slave read. This is so that any snoop push writes won't get backed up behind the write the the expansion bridge (which, in turn, could be blocked by a read on PCI, etc...)

The last term incorporates all of the retry components into DoARtry_. ARtry is DoARtry delayed by one register delay.

The following Table maps the deadlock rules identified in the DETAILED DESCRIPTION to the deadlock cases below:

Deadlock Rule (Description)	Verilog Deadlock case
A1	ExRdRetry
A2	L2Retry
A3	ExWrRetry
B4	ExRdRetry

Deadlock Rule (Description)	Verilog Deadlock case
B7	ExCrossRdRetry (1,4)
B8	ExCrossRdRetry (2,3)

Code

```
assign #'AD L2Retry          = (!sackIn_[0] || !sackIn_[1]) && L2Cares &&
                               ! (CpuMemWr && !DisableDBWO && ! SnoopQFull) &&
                               | (~MasNum & ValidToEx);

assign #'AD TransFullRetry   = TransFull && DatSack && !DataDone;

assign #'AD ExRdRetry        = (!aackIn_ || !aackOut_) &&
                               ! (CpuMemWr && !DisableDBWO && !SnoopQFull) &&
                               (    (qSackIn_[4] && | (~MasNum & ValidEx1Rd))
                                 || (qSackIn_[5] && | (~MasNum & ValidEx2Rd))  );

assign #'AD ExCrossRdRetry = oldTT1 &&
                          (      ( Ex1HasSlvRd && !MasNum[1] && !qSackIn_[5])  // B1 -> B2 (1)
                              ||  ( Ex1HasSlvRd && !MasNum[2] && !qSackIn_[4])  // B2 -> B1 (2)
                              ||  ( Ex2HasSlvRd && !MasNum[1] && !qSackIn_[5])  // B1 -> B2 (3)
                              ||  ( Ex2HasSlvRd && !MasNum[2] && !qSackIn_[4])  // B2 -> B1 (4)
                          );

assign #'AD ExWrRetry = !oldTT1 && (!MasNum[3] || !MasNum[4]) &&
                          (      ( Ex1HasSlvRd && !qSackIn_[4])
                              ||  ( Ex2HasSlvRd && !qSackIn_[5])
                          );

assign #'AD DoArtry_         = !    ( oldTT3 &&
                                          (     L2Retry
                                             || TransFullRetry
                                             || ExRdRetry
                                             || ExCrossRdRetry
                                             || ExWrRetry
                                          )
                                    );
```

```
output  [0:4]   SlvMatch0, SlvMatch1, SlvMatch2;
output  [0:4]   SlvRdReady0, SlvRdReady1, SlvRdReady2;

input   [0:4]   DOutSQ0_0, DOutSQ0_1, DOutSQ0_2,
                DOutSQ1_0, DOutSQ1_1, DOutSQ1_2,
                DOutSQ2_0, DOutSQ2_1, DOutSQ2_2,
                DOutSQ3_0, DOutSQ3_1, DOutSQ3_2,
                DOutSQ4_0, DOutSQ4_1, DOutSQ4_2,
                DOutSQ5_0, DOutSQ5_1, DOutSQ5_2,
                DOutSQ6_0, DOutSQ6_1, DOutSQ6_2;
input   [0:2]   SQValid0, SQValid1, SQValid2, SQValid3,
                SQValid4, SQValid5, SQValid6;
input   [0:6]   rdda0In_, rdda1In_, rdda2In_;
input   [0:6]   DOutMQ0, DOutMQ1, DOutMQ2, DOutMQ3, DOutMQ4;
input   [0:6]   AddrHit01, AddrHit02, AddrHit12;


reg     [0:4]   SlvMatch0, SlvMatch1, SlvMatch2;
reg     [0:4]   SlvRdReady0, SlvRdReady1, SlvRdReady2;

wire [0:6] FromNode0_0_ = {     DOutSQ0_0[0],DOutSQ1_0[0],DOutSQ2_0[0],DOutSQ3_0[0],
                                DOutSQ4_0[0],DOutSQ5_0[0],DOutSQ6_0[0]};
wire [0:6] FromNode1_0_ = {     DOutSQ0_0[1],DOutSQ1_0[1],DOutSQ2_0[1],DOutSQ3_0[1],
                                DOutSQ4_0[1],DOutSQ5_0[1],DOutSQ6_0[1]};
wire [0:6] FromNode2_0_ = {     DOutSQ0_0[2],DOutSQ1_0[2],DOutSQ2_0[2],DOutSQ3_0[2],
                                DOutSQ4_0[2],DOutSQ5_0[2],DOutSQ6_0[2]};
wire [0:6] FromNode3_0_ = {     DOutSQ0_0[3],DOutSQ1_0[3],DOutSQ2_0[3],DOutSQ3_0[3],
                                DOutSQ4_0[3],DOutSQ5_0[3],DOutSQ6_0[3]};
wire [0:6] FromNode4_0_ = {     DOutSQ0_0[4],DOutSQ1_0[4],DOutSQ2_0[4],DOutSQ3_0[4],
                                DOutSQ4_0[4],DOutSQ5_0[4],DOutSQ6_0[4]};

wire [0:6] FromNode0_1_ = {     DOutSQ0_0[0],DOutSQ1_0[0],DOutSQ2_0[0],DOutSQ3_0[0],
                                DOutSQ4_0[0],DOutSQ5_0[0],DOutSQ6_0[0]};
wire [0:6] FromNode1_1_ = {     DOutSQ0_0[1],DOutSQ1_0[1],DOutSQ2_0[1],DOutSQ3_0[1],
                                DOutSQ4_0[1],DOutSQ5_0[1],DOutSQ6_0[1]};
wire [0:6] FromNode2_1_ = {     DOutSQ0_0[2],DOutSQ1_0[2],DOutSQ2_0[2],DOutSQ3_0[2],
                                DOutSQ4_0[2],DOutSQ5_0[2],DOutSQ6_0[2]};
wire [0:6] FromNode3_1_ = {     DOutSQ0_0[3],DOutSQ1_0[3],DOutSQ2_0[3],DOutSQ3_0[3],
                                DOutSQ4_0[3],DOutSQ5_0[3],DOutSQ6_0[3]};
wire [0:6] FromNode4_1_ = {     DOutSQ0_0[4],DOutSQ1_0[4],DOutSQ2_0[4],DOutSQ3_0[4],
                                DOutSQ4_0[4],DOutSQ5_0[4],DOutSQ6_0[4]};

wire [0:6] FromNode0_2_ = {     DOutSQ0_0[0],DOutSQ1_0[0],DOutSQ2_0[0],DOutSQ3_0[0],
                                DOutSQ4_0[0],DOutSQ5_0[0],DOutSQ6_0[0]};
wire [0:6] FromNode1_2_ = {     DOutSQ0_0[1],DOutSQ1_0[1],DOutSQ2_0[1],DOutSQ3_0[1],
                                DOutSQ4_0[1],DOutSQ5_0[1],DOutSQ6_0[1]};
wire [0:6] FromNode2_2_ = {     DOutSQ0_0[2],DOutSQ1_0[2],DOutSQ2_0[2],DOutSQ3_0[2],
                                DOutSQ4_0[2],DOutSQ5_0[2],DOutSQ6_0[2]};
wire [0:6] FromNode3_2_ = {     DOutSQ0_0[3],DOutSQ1_0[3],DOutSQ2_0[3],DOutSQ3_0[3],
                                DOutSQ4_0[3],DOutSQ5_0[3],DOutSQ6_0[3]};
wire [0:6] FromNode4_2_ = {     DOutSQ0_0[4],DOutSQ1_0[4],DOutSQ2_0[4],DOutSQ3_0[4],
                                DOutSQ4_0[4],DOutSQ5_0[4],DOutSQ6_0[4]};


wire    [0:6]   SlvValid0 = {   SQValid0[0],SQValid1[0],SQValid2[0],SQValid3[0],
                                SQValid4[0],SQValid5[0],SQValid6[0]};
wire    [0:6]   SlvValid1 = {   SQValid0[1],SQValid1[1],SQValid2[1],SQValid3[1],
                                SQValid4[1],SQValid5[1],SQValid6[1]};
wire    [0:6]   SlvValid2 = {   SQValid0[2],SQValid1[2],SQValid2[2],SQValid3[2],
                                SQValid4[2],SQValid5[2],SQValid6[2]};


wire    [0:6]   ValidFrom0_0_, ValidFrom1_0_, ValidFrom2_0_, ValidFrom3_0_, ValidFrom4_0_;
wire    [0:6]   ValidFrom0_1_, ValidFrom1_1_, ValidFrom2_1_, ValidFrom3_1_, ValidFrom4_1_;
```

```
wire    [0:6]   ValidFrom0_2_, ValidFrom1_2_, ValidFrom2_2_, ValidFrom3_2_, ValidFrom4_2_;


assign #1       ValidFrom0_0_ = FromNode0_0_ | SlvValid0;
assign #1       ValidFrom1_0_ = FromNode1_0_ | SlvValid0;
assign #1       ValidFrom2_0_ = FromNode2_0_ | SlvValid0;
assign #1       ValidFrom3_0_ = FromNode3_0_ | SlvValid0;
assign #1       ValidFrom4_0_ = FromNode4_0_ | SlvValid0;

assign #1       ValidFrom0_1_ = FromNode0_1_ | SlvValid1;
assign #1       ValidFrom1_1_ = FromNode1_1_ | SlvValid1;
assign #1       ValidFrom2_1_ = FromNode2_1_ | SlvValid1;
assign #1       ValidFrom3_1_ = FromNode3_1_ | SlvValid1;
assign #1       ValidFrom4_1_ = FromNode4_1_ | SlvValid1;

assign #1       ValidFrom0_2_ = FromNode0_2_ | SlvValid2;
assign #1       ValidFrom1_2_ = FromNode1_2_ | SlvValid2;
assign #1       ValidFrom2_2_ = FromNode2_2_ | SlvValid2;
assign #1       ValidFrom3_2_ = FromNode3_2_ | SlvValid2;
assign #1       ValidFrom4_2_ = FromNode4_2_ | SlvValid2;



always @(DOutMQ0 or ValidFrom0_0_ or ValidFrom0_1_ or ValidFrom0_2_ or
         rdda0In_ or rdda1In_ or rdda2In_ or AddrHit01 or AddrHit02 or AddrHit12 ) begin
        casex (DOutMQ0[0:6])     // synopsys full_case parallel_case
          7'b0xxxxxx: begin
                SlvMatch0[0]    <= #1 ~ValidFrom0_0_[0];
                SlvMatch1[0]    <= #1 ~ValidFrom0_1_[0] && !AddrHit01[0];
                SlvMatch2[0]    <= #1 ~ValidFrom0_2_[0] && !AddrHit02[0] && !AddrHit12[0];
                SlvRdReady0[0] <= #1 ~rdda0In_[0];
                SlvRdReady1[0] <= #1 ~rdda1In_[0];
                SlvRdReady2[0] <= #1 ~rdda2In_[0];
          end
          7'b10xxxxx: begin
                SlvMatch0[0]    <= #1 ~ValidFrom0_0_[1];
                SlvMatch1[0]    <= #1 ~ValidFrom0_1_[1] && !AddrHit01[1];
                SlvMatch2[0]    <= #1 ~ValidFrom0_2_[1] && !AddrHit02[1] && !AddrHit12[1];
                SlvRdReady0[0] <= #1 ~rdda0In_[1];
                SlvRdReady1[0] <= #1 ~rdda1In_[1];
                SlvRdReady2[0] <= #1 ~rdda2In_[1];
          end
          7'b110xxxx: begin
                SlvMatch0[0]    <= #1 ~ValidFrom0_0_[2];
                SlvMatch1[0]    <= #1 ~ValidFrom0_1_[2] && !AddrHit01[2];
                SlvMatch2[0]    <= #1 ~ValidFrom0_2_[2] && !AddrHit02[2] && !AddrHit12[2];
                SlvRdReady0[0] <= #1 ~rdda0In_[2];
                SlvRdReady1[0] <= #1 ~rdda1In_[2];
                SlvRdReady2[0] <= #1 ~rdda2In_[2];
          end
          7'b1110xxx: begin
                SlvMatch0[0]    <= #1 ~ValidFrom0_0_[3];
                SlvMatch1[0]    <= #1 ~ValidFrom0_1_[3] && !AddrHit01[3];
                SlvMatch2[0]    <= #1 ~ValidFrom0_2_[3] && !AddrHit02[3] && !AddrHit12[3];
                SlvRdReady0[0] <= #1 ~rdda0In_[3];
                SlvRdReady1[0] <= #1 ~rdda1In_[3];
                SlvRdReady2[0] <= #1 ~rdda2In_[3];
          end
          7'b11110xx: begin
                SlvMatch0[0]    <= #1 ~ValidFrom0_0_[4];
                SlvMatch1[0]    <= #1 ~ValidFrom0_1_[4] && !AddrHit01[4];
                SlvMatch2[0]    <= #1 ~ValidFrom0_2_[4] && !AddrHit02[4] && !AddrHit12[4];
                SlvRdReady0[0] <= #1 ~rdda0In_[4];
```

```
            SlvRdReady1[0] <= #1 ~rdda1In_[4];
            SlvRdReady2[0] <= #1 ~rdda2In_[4];
        end
        7'b111110x: begin
            SlvMatch0[0]   <= #1 ~ValidFrom0_0_[5];
            SlvMatch1[0]   <= #1 ~ValidFrom0_1_[5] && !AddrHit01[5];
            SlvMatch2[0]   <= #1 ~ValidFrom0_2_[5] && !AddrHit02[5] && !AddrHit12[5];
            SlvRdReady0[0] <= #1 ~rdda0In_[5];
            SlvRdReady1[0] <= #1 ~rdda1In_[5];
            SlvRdReady2[0] <= #1 ~rdda2In_[5];
        end
        7'b1111110: begin
            SlvMatch0[0]   <= #1 ~ValidFrom0_0_[6];
            SlvMatch1[0]   <= #1 ~ValidFrom0_1_[6] && !AddrHit01[6];
            SlvMatch2[0]   <= #1 ~ValidFrom0_2_[6] && !AddrHit02[6] && !AddrHit12[6];
            SlvRdReady0[0] <= #1 ~rdda0In_[6];
            SlvRdReady1[0] <= #1 ~rdda1In_[6];
            SlvRdReady2[0] <= #1 ~rdda2In_[6];
        end
        7'b1111111: begin
            SlvMatch0[0]   <= #1 1'b0;
            SlvMatch1[0]   <= #1 1'b0;
            SlvMatch2[0]   <= #1 1'b0;
            SlvRdReady0[0] <= #1 1'b0;
            SlvRdReady1[0] <= #1 1'b0;
            SlvRdReady2[0] <= #1 1'b0;
        end
      endcase
end


always @(DOutMQ1 or ValidFrom0_0_ or ValidFrom0_1_ or ValidFrom0_2_ or
        rdda0In_ or rdda1In_ or rdda2In_) begin
      casex (DOutMQ1[0:6])     // synopsys full_case parallel_case
        7'b0xxxxxx: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[0];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[0] && !AddrHit01[0];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[0] && !AddrHit02[0] && !AddrHit12[0];
            SlvRdReady0[1] <= #1 ~rdda0In_[0];
            SlvRdReady1[1] <= #1 ~rdda1In_[0];
            SlvRdReady2[1] <= #1 ~rdda2In_[0];
        end
        7'b10xxxxx: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[1];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[1] && !AddrHit01[1];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[1] && !AddrHit02[1] && !AddrHit12[1];
            SlvRdReady0[1] <= #1 ~rdda0In_[1];
            SlvRdReady1[1] <= #1 ~rdda1In_[1];
            SlvRdReady2[1] <= #1 ~rdda2In_[1];
        end
        7'b110xxxx: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[2];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[2] && !AddrHit01[2];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[2] && !AddrHit02[2] && !AddrHit12[2];
            SlvRdReady0[1] <= #1 ~rdda0In_[2];
            SlvRdReady1[1] <= #1 ~rdda1In_[2];
            SlvRdReady2[1] <= #1 ~rdda2In_[2];
        end
        7'b1110xxx: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[3];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[3] && !AddrHit01[3];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[3] && !AddrHit02[3] && !AddrHit12[3];
            SlvRdReady0[1] <= #1 ~rdda0In_[3];
```

```
            SlvRdReady1[1] <= #1 ~rdda1In_[3];
            SlvRdReady2[1] <= #1 ~rdda2In_[3];
          end
          7'b11110xx: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[4];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[4] && !AddrHit01[4];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[4] && !AddrHit02[4] && !AddrHit12[4];
            SlvRdReady0[1] <= #1 ~rdda0In_[4];
            SlvRdReady1[1] <= #1 ~rdda1In_[4];
            SlvRdReady2[1] <= #1 ~rdda2In_[4];
          end
          7'b1111110x: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[5];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[5] && !AddrHit01[5];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[5] && !AddrHit02[5] && !AddrHit12[5];
            SlvRdReady0[1] <= #1 ~rdda0In_[5];
            SlvRdReady1[1] <= #1 ~rdda1In_[5];
            SlvRdReady2[1] <= #1 ~rdda2In_[5];
          end
          7'b1111110: begin
            SlvMatch0[1]   <= #1 ~ValidFrom0_0_[6];
            SlvMatch1[1]   <= #1 ~ValidFrom0_1_[6] && !AddrHit01[6];
            SlvMatch2[1]   <= #1 ~ValidFrom0_2_[6] && !AddrHit02[6] && !AddrHit12[6];
            SlvRdReady0[1] <= #1 ~rdda0In_[6];
            SlvRdReady1[1] <= #1 ~rdda1In_[6];
            SlvRdReady2[1] <= #1 ~rdda2In_[6];
          end
          7'b1111111: begin
            SlvMatch0[1]   <= #1 1'b0;
            SlvMatch1[1]   <= #1 1'b0;
            SlvMatch2[1]   <= #1 1'b0;
            SlvRdReady0[1] <= #1 1'b0;
            SlvRdReady1[1] <= #1 1'b0;
            SlvRdReady2[1] <= #1 1'b0;
          end
        endcase
end


always @(DOutMQ2 or ValidFrom0_0_ or ValidFrom0_1_ or ValidFrom0_2_ or
         rdda0In_ or rdda1In_ or rdda2In_) begin
        casex (DOutMQ2[0:6])     // synopsys full_case parallel_case
          7'b0xxxxxx: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[0];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[0] && !AddrHit01[0];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[0] && !AddrHit02[0] && !AddrHit12[0];
            SlvRdReady0[2] <= #1 ~rdda0In_[0];
            SlvRdReady1[2] <= #1 ~rdda1In_[0];
            SlvRdReady2[2] <= #1 ~rdda2In_[0];
          end
          7'b10xxxxx: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[1];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[1] && !AddrHit01[1];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[1] && !AddrHit02[1] && !AddrHit12[1];
            SlvRdReady0[2] <= #1 ~rdda0In_[1];
            SlvRdReady1[2] <= #1 ~rdda1In_[1];
            SlvRdReady2[2] <= #1 ~rdda2In_[1];
          end
          7'b110xxxx: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[2];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[2] && !AddrHit01[2];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[2] && !AddrHit02[2] && !AddrHit12[2];
            SlvRdReady0[2] <= #1 ~rdda0In_[2];
```

```
            SlvRdReady1[2] <= #1 ~rdda1In_[2];
            SlvRdReady2[2] <= #1 ~rdda2In_[2];
          end
          7'b1110xxx: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[3];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[3] && !AddrHit01[3];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[3] && !AddrHit02[3] && !AddrHit12[3];
            SlvRdReady0[2] <= #1 ~rdda0In_[3];
            SlvRdReady1[2] <= #1 ~rdda1In_[3];
            SlvRdReady2[2] <= #1 ~rdda2In_[3];
          end
          7'b11110xx: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[4];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[4] && !AddrHit01[4];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[4] && !AddrHit02[4] && !AddrHit12[4];
            SlvRdReady0[2] <= #1 ~rdda0In_[4];
            SlvRdReady1[2] <= #1 ~rdda1In_[4];
            SlvRdReady2[2] <= #1 ~rdda2In_[4];
          end
          7'b111110x: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[5];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[5] && !AddrHit01[5];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[5] && !AddrHit02[5] && !AddrHit12[5];
            SlvRdReady0[2] <= #1 ~rdda0In_[5];
            SlvRdReady1[2] <= #1 ~rdda1In_[5];
            SlvRdReady2[2] <= #1 ~rdda2In_[5];
          end
          7'b1111110: begin
            SlvMatch0[2]   <= #1 ~ValidFrom0_0_[6];
            SlvMatch1[2]   <= #1 ~ValidFrom0_1_[6] && !AddrHit01[6];
            SlvMatch2[2]   <= #1 ~ValidFrom0_2_[6] && !AddrHit02[6] && !AddrHit12[6];
            SlvRdReady0[2] <= #1 ~rdda0In_[6];
            SlvRdReady1[2] <= #1 ~rdda1In_[6];
            SlvRdReady2[2] <= #1 ~rdda2In_[6];
          end
          7'b1111111: begin
            SlvMatch0[2]   <= #1 1'b0;
            SlvMatch1[2]   <= #1 1'b0;
            SlvMatch2[2]   <= #1 1'b0;
            SlvRdReady0[2] <= #1 1'b0;
            SlvRdReady1[2] <= #1 1'b0;
            SlvRdReady2[2] <= #1 1'b0;
          end
        endcase
end


always @(DOutMQ3 or ValidFrom0_0_ or ValidFrom0_1_ or ValidFrom0_2_ or
         rdda0In_ or rdda1In_ or rdda2In_) begin
        casex (DOutMQ3[0:6])     // synopsys full_case parallel_case
          7'b0xxxxxx: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[0];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[0] && !AddrHit01[0];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[0] && !AddrHit02[0] && !AddrHit12[0];
            SlvRdReady0[3] <= #1 ~rdda0In_[0];
            SlvRdReady1[3] <= #1 ~rdda1In_[0];
            SlvRdReady2[3] <= #1 ~rdda2In_[0];
          end
          7'b10xxxxx: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[1];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[1] && !AddrHit01[1];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[1] && !AddrHit02[1] && !AddrHit12[1];
            SlvRdReady0[3] <= #1 ~rdda0In_[1];
```

```
            SlvRdReady1[3] <= #1 ~rdda1In_[1];
            SlvRdReady2[3] <= #1 ~rdda2In_[1];
          end
          7'b110xxxx: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[2];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[2] && !AddrHit01[2];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[2] && !AddrHit02[2] && !AddrHit12[2];
            SlvRdReady0[3] <= #1 ~rdda0In_[2];
            SlvRdReady1[3] <= #1 ~rdda1In_[2];
            SlvRdReady2[3] <= #1 ~rdda2In_[2];
          end
          7'b1110xxx: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[3];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[3] && !AddrHit01[3];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[3] && !AddrHit02[3] && !AddrHit12[3];
            SlvRdReady0[3] <= #1 ~rdda0In_[3];
            SlvRdReady1[3] <= #1 ~rdda1In_[3];
            SlvRdReady2[3] <= #1 ~rdda2In_[3];
          end
          7'b11110xx: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[4];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[4] && !AddrHit01[4];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[4] && !AddrHit02[4] && !AddrHit12[4];
            SlvRdReady0[3] <= #1 ~rdda0In_[4];
            SlvRdReady1[3] <= #1 ~rdda1In_[4];
            SlvRdReady2[3] <= #1 ~rdda2In_[4];
          end
          7'b111110x: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[5];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[5] && !AddrHit01[5];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[5] && !AddrHit02[5] && !AddrHit12[5];
            SlvRdReady0[3] <= #1 ~rdda0In_[5];
            SlvRdReady1[3] <= #1 ~rdda1In_[5];
            SlvRdReady2[3] <= #1 ~rdda2In_[5];
          end
          7'b1111110: begin
            SlvMatch0[3]   <= #1 ~ValidFrom0_0_[6];
            SlvMatch1[3]   <= #1 ~ValidFrom0_1_[6] && !AddrHit01[6];
            SlvMatch2[3]   <= #1 ~ValidFrom0_2_[6] && !AddrHit02[6] && !AddrHit12[6];
            SlvRdReady0[3] <= #1 ~rdda0In_[6];
            SlvRdReady1[3] <= #1 ~rdda1In_[6];
            SlvRdReady2[3] <= #1 ~rdda2In_[6];
          end
          7'b1111111: begin
            SlvMatch0[3]   <= #1 1'b0;
            SlvMatch1[3]   <= #1 1'b0;
            SlvMatch2[3]   <= #1 1'b0;
            SlvRdReady0[3] <= #1 1'b0;
            SlvRdReady1[3] <= #1 1'b0;
            SlvRdReady2[3] <= #1 1'b0;
          end
        endcase
end


always @(DOutMQ4 or ValidFrom0_0_ or ValidFrom0_1_ or ValidFrom0_2_ or
         rdda0In_ or rdda1In_ or rdda2In_) begin
        casex (DOutMQ4[0:6])     // synopsys full_case parallel_case
          7'b0xxxxxx: begin
            SlvMatch0[4]   <= #1 ~ValidFrom0_0_[0];
            SlvMatch1[4]   <= #1 ~ValidFrom0_1_[0] && !AddrHit01[0];
            SlvMatch2[4]   <= #1 ~ValidFrom0_2_[0] && !AddrHit02[0] && !AddrHit12[0];
            SlvRdReady0[4] <= #1 ~rdda0In_[0];
```

```
                SlvRdReady1[4] <= #1 ~rdda1In_[0];
                SlvRdReady2[4] <= #1 ~rdda2In_[0];
            end
            7'b10xxxxx: begin
                SlvMatch0[4]   <= #1 ~ValidFrom0_0_[1];
                SlvMatch1[4]   <= #1 ~ValidFrom0_1_[1] && !AddrHit01[1];
                SlvMatch2[4]   <= #1 ~ValidFrom0_2_[1] && !AddrHit02[1] && !AddrHit12[1];
                SlvRdReady0[4] <= #1 ~rdda0In_[1];
                SlvRdReady1[4] <= #1 ~rdda1In_[1];
                SlvRdReady2[4] <= #1 ~rdda2In_[1];
            end
            7'b110xxxx: begin
                SlvMatch0[4]   <= #1 ~ValidFrom0_0_[2];
                SlvMatch1[4]   <= #1 ~ValidFrom0_1_[2] && !AddrHit01[2];
                SlvMatch2[4]   <= #1 ~ValidFrom0_2_[2] && !AddrHit02[2] && !AddrHit12[2];
                SlvRdReady0[4] <= #1 ~rdda0In_[2];
                SlvRdReady1[4] <= #1 ~rdda1In_[2];
                SlvRdReady2[4] <= #1 ~rdda2In_[2];
            end
            7'b1110xxx: begin
                SlvMatch0[4]   <= #1 ~ValidFrom0_0_[3];
                SlvMatch1[4]   <= #1 ~ValidFrom0_1_[3] && !AddrHit01[3];
                SlvMatch2[4]   <= #1 ~ValidFrom0_2_[3] && !AddrHit02[3] && !AddrHit12[3];
                SlvRdReady0[4] <= #1 ~rdda0In_[3];
                SlvRdReady1[4] <= #1 ~rdda1In_[3];
                SlvRdReady2[4] <= #1 ~rdda2In_[3];
            end
            7'b11110xx: begin
                SlvMatch0[4]   <= #1 ~ValidFrom0_0_[4];
                SlvMatch1[4]   <= #1 ~ValidFrom0_1_[4] && !AddrHit01[4];
                SlvMatch2[4]   <= #1 ~ValidFrom0_2_[4] && !AddrHit02[4] && !AddrHit12[4];
                SlvRdReady0[4] <= #1 ~rdda0In_[4];
                SlvRdReady1[4] <= #1 ~rdda1In_[4];
                SlvRdReady2[4] <= #1 ~rdda2In_[4];
            end
            7'b111110x: begin
                SlvMatch0[4]   <= #1 ~ValidFrom0_0_[5];
                SlvMatch1[4]   <= #1 ~ValidFrom0_1_[5] && !AddrHit01[5];
                SlvMatch2[4]   <= #1 ~ValidFrom0_2_[5] && !AddrHit02[5] && !AddrHit12[5];
                SlvRdReady0[4] <= #1 ~rdda0In_[5];
                SlvRdReady1[4] <= #1 ~rdda1In_[5];
                SlvRdReady2[4] <= #1 ~rdda2In_[5];
            end
            7'b1111110: begin
                SlvMatch0[4]   <= #1 ~ValidFrom0_0_[6];
                SlvMatch1[4]   <= #1 ~ValidFrom0_1_[6] && !AddrHit01[6];
                SlvMatch2[4]   <= #1 ~ValidFrom0_2_[6] && !AddrHit02[6] && !AddrHit12[6];
                SlvRdReady0[4] <= #1 ~rdda0In_[6];
                SlvRdReady1[4] <= #1 ~rdda1In_[6];
                SlvRdReady2[4] <= #1 ~rdda2In_[6];
            end
            7'b1111111: begin
                SlvMatch0[4]   <= #1 1'b0;
                SlvMatch1[4]   <= #1 1'b0;
                SlvMatch2[4]   <= #1 1'b0;
                SlvRdReady0[4] <= #1 1'b0;
                SlvRdReady1[4] <= #1 1'b0;
                SlvRdReady2[4] <= #1 1'b0;
            end
        endcase
    end
```

```
output  [0:4]   dbgOut_;
output  [0:6]   ssd0Out_, ssd1Out_, ssd2Out_;

input   [0:4]   MasNum_;
input   [0:8]   DOutMQ0,
                DOutMQ1,
                DOutMQ2,
                DOutMQ3,
                DOutMQ4;
input   [0:4]   MQEmpty;
input   [0:4]   SlvMatch0;
input   [0:4]   SlvMatch1;
input   [0:4]   SlvMatch2;
input   [0:4]   SlvRdReady0;
input   [0:4]   SlvRdReady1;
input   [0:4]   SlvRdReady2;
input   [0:4]   PageHit01;      // Slave-based page hits mapped to masters
input   [0:4]   PageHit02;
input   [0:4]   PageHit12;
input           Clk;
input           Reset_;


reg     [0:4]   CalcDBG;
reg     [0:6]   CalcSSD0;
reg     [0:6]   CalcSSD1;
reg     [0:6]   CalcSSD2;

wire    [0:4]   MasReady0;
wire    [0:4]   MasReady1;
wire    [0:4]   MasReady2;

wire    [0:4]   MasReady;

wire            DBGPend;

wire    [0:4]   ReadOp = {      DOutMQ0[8],
                                DOutMQ1[8],
                                DOutMQ2[8],
                                DOutMQ3[8],
                                DOutMQ4[8]  };


assign #`AD     MasReady0 = SlvMatch0 & ~MQEmpty & (~ReadOp | SlvRdReady0);
assign #`AD     MasReady1 = SlvMatch1 & ~MQEmpty & (~ReadOp | SlvRdReady1);
assign #`AD     MasReady2 = SlvMatch2 & ~MQEmpty & (~ReadOp | SlvRdReady2);

assign #`AD     MasReady = MasReady0 | MasReady1 | MasReady2;

assign #`AD     DBGPend = |MasReady;


wire    [0:4]   Choose0 =  MasReady0;
wire    [0:4]   Choose1 = ~MasReady0 &  MasReady1
                        |  PageHit01 &  MasReady1;
wire    [0:4]   Choose2 = ~MasReady0 & ~MasReady1 & MasReady2
                        |  PageHit02 & ~MasReady1 & MasReady2
                        |  PageHit12 &  MasReady2;

/*
```

```
 */

always @(SnoopDOut or MasReady or Choose0 or Choose1 or Choose2 or
          DOutMQ0 or DOutMQ1 or DOutMQ2 or DOutMQ3 or DOutMQ4) begin

        casex (MasReady)                    /* synopsys full_case parallel_case */
        5'b1xxxx: begin
                CalcDBG  <= #`AD 5'b01111;
                CalcSSD0 <= #`AD Choose0[0] ? DOutMQ0[0:6] : 7'b1111111;
                CalcSSD1 <= #`AD Choose1[0] ? DOutMQ0[0:6] : 7'b1111111;
                CalcSSD2 <= #`AD Choose2[0] ? DOutMQ0[0:6] : 7'b1111111;
        end
        5'b01xxx: begin
                CalcDBG  <= #`AD 5'b10111;
                CalcSSD0 <= #`AD Choose0[1] ? DOutMQ1[0:6] : 7'b1111111;
                CalcSSD1 <= #`AD Choose1[1] ? DOutMQ1[0:6] : 7'b1111111;
                CalcSSD2 <= #`AD Choose2[1] ? DOutMQ1[0:6] : 7'b1111111;
        end
        5'b001xx: begin
                CalcDBG  <= #`AD 5'b11011;
                CalcSSD0 <= #`AD Choose0[2] ? DOutMQ2[0:6] : 7'b1111111;
                CalcSSD1 <= #`AD Choose1[2] ? DOutMQ2[0:6] : 7'b1111111;
                CalcSSD2 <= #`AD Choose2[2] ? DOutMQ2[0:6] : 7'b1111111;
        end
        5'b0001x: begin
                CalcDBG  <= #`AD 5'b11101;
                CalcSSD0 <= #`AD Choose0[3] ? DOutMQ3[0:6] : 7'b1111111;
                CalcSSD1 <= #`AD Choose1[3] ? DOutMQ3[0:6] : 7'b1111111;
                CalcSSD2 <= #`AD Choose2[3] ? DOutMQ3[0:6] : 7'b1111111;
        end
        5'b00001: begin
                CalcDBG  <= #`AD 5'b11110;
                CalcSSD0 <= #`AD Choose0[4] ? DOutMQ4[0:6] : 7'b1111111;
                CalcSSD1 <= #`AD Choose1[4] ? DOutMQ4[0:6] : 7'b1111111;
                CalcSSD2 <= #`AD Choose2[4] ? DOutMQ4[0:6] : 7'b1111111;
        end
        5'b00000: begin
                CalcDBG  <= #`AD 5'b11111;
                CalcSSD0 <= #`AD 7'b1111111;
        end
        endcase
end

...   ...   ...
...   ...   ...
...   ...   ...

(next dbgOut_[0:4] is CalcDBG[0:4])
(next ssd0Out_[0:6] is CalcSSD0[0:6])
(next ssd1Out_[0:6] is CalcSSD1[0:6])
(next ssd2Out_[0:6] is CalcSSD2[0:6])
```

```
/*
 *      Retry generation
 *
 *      The first term, TransFullRetry, kills an access when we already have
 *      the maximum number of outstanding transactions (3).  This term comes
 *      from FM_ArbDatSM.
 *
 *      The second term, BriRdRetry, is an OR of a vector showing an AAck of
 *      a master that has an outstanding Bridge read.  This transaction must be
 *      to a non-Bridge slave, and only writes to memory are excepted.
 *
 *      The third term, BriCrossRdRetry, will kill a Bridge's master read of
 *      the other Bridge if the Bridge master already has a slave read outstanding
 *      to it, OR if the Bridge slave already has an outstanding read.
 *
 *      The fourth term, BriWrRetry, will kill a CPU write to a Bridge
 *      that has an outstanding slave read.  This is so that any snoop push
 *      writes won't get backed up behind the write to Bridge (which, in turn,
 *      could be blocked by a read on PCI, etc...).
 *
 *      The fifth term, SlvRetry, comes from FM_ARBusSlave, and indicates that
 *      a slave is AAck'ing a transaction, but its address queue is full.
 *      This is modal behavior, enabled by EnQFullRetry.  If not enabled,
 *      FM_ARBusSlave will simply hold off AAck until the slave queue is no
 *      longer empty (the default behavior).
 *
 *      The sixth term, VidRetry, will retry a Video Bridge  write to a destination
 *      other than a Bridge it's currently writing.
 *
 *      The seventh and eigth terms, Bri[01]MultiWrRetry, will kill a write from
 *      a Bridge if that Bridge already has a write outstanding to the other bridge.
 *      Multiple writes to the same bridge are OK - just not multiple writes to
 *      multiple slaves.
 *
 *      The last term incorporates all of the retry components into DoARtry_.
 */

assign #`AD BriRdRetry      = (!aackIn_ || !aackOut_) &&
                              !(CpuMemWr && !DisableDBWO) &&
                              (   (qSackIn_[5] && |(~MasNum_ & ValidBri1Rd))
                               ||(qSackIn_[6] && |(~MasNum_ & ValidBri2Rd))  );

assign #`AD BriCrossRdRetry = //oldTT1 &&
                              (   ( Bri1HasSlvRd && !MasNum_[1] && !qSackIn_[6])  // B1 -> B2
                               || ( Bri1HasSlvRd && !MasNum_[2] && !qSackIn_[5])  // B2 -> B1
                               || ( Bri2HasSlvRd && !MasNum_[1] && !qSackIn_[6])  // B1 -> B2
                               || ( Bri2HasSlvRd && !MasNum_[2] && !qSackIn_[5])  // B2 -> B1
                              );

assign #`AD BriWrRetry = !oldTT1 && (!MasNum_[3] || !MasNum_[4]) &&
                         (   ( Bri1HasSlvRd && !qSackIn_[5])
                          || ( Bri2HasSlvRd && !qSackIn_[6])
                         );

assign #`AD VidRetry = !oldTT1 && !MasNum_[0] && (!aackIn_ || !aackOut_) &&
                       (   ( ValidBri1Wr[0] && qSackIn_[5] )
                        || ( ValidBri2Wr[0] && qSackIn_[6] )
                       );

assign #`AD Bri0MultiWrRetry = !oldTT1 && !MasNum_[1] && (!aackIn_ || !aackOut_) &&
                       ( ( ValidVidWr[1]  && qSackIn_[4] )
                       ||( ValidBri2Wr[1] && qSackIn_[6] ) );
```

```
assign #`AD BrilMultiWrRetry = !oldTT1 && !MasNum_[2] && (!aackIn_ || !aackOut_) &&
                        ( ( ValidVidWr[2]  && qSackIn_[4] )
                        ||( ValidBrilWr[2] && qSackIn_[5] ) );

assign #`AD DoArtry_       = !  ( oldTT3 &&
                                  (    TransFullRetry
                                    || BriRdRetry
                                    || BriCrossRdRetry
                                    || BriWrRetry
                                    || VidRetry
                                    || SlvRetry
                                    || Bri0MultiWrRetry
                                    || BrilMultiWrRetry
                                  )
                              );
```